ModusToolbox™ Machine Learning user guide

ModusToolbox™ tools package version 3.7.0 or later

Machine Learning pack version 3.1.0

About this document

Scope and purpose

This document helps you understand all the various aspects of the ModusToolbox™ Machine Learning (ML) solution.

Document conventions

Convention	Explanation
Bold	Emphasizes heading levels, column headings, menus and sub-menus
Italics	Denotes file names and paths.
`Courier New`	Denotes APIs, functions, interrupt handlers, events, data types, error handlers, file/folder names, directories, command line inputs, code snippets
File > New	Indicates that a cascading sub-menu opens when you select a menu item

Abbreviations and definitions

The following define the abbreviations and terms used in this document:

- ML – Machine Learning
- NN –neural network
- NPZ – NumPy array in zipped format [1]
- RNN – Recurrent neural network
- TFLM – TensorFlow Lite for microcontrollers
- NPU – Neural Processing Unit

References

Refer to the following documents and websites for more information as needed:

- ModusToolbox™ tools package user guide
- ModusToolbox™ Machine Learning Configurator guide
- Machine Learning on the edge application note
- https://github.com/infineon/ml-middleware
- https://github.com/infineon/ml-tflite-micro

[1] When unzipped, it provides validation data as NumPy arrays used by the tool.

About this document

- https://www.infineon.com/cms/en/design-support/tools/sdk/modustoolbox-software/modustoolbox-machine-learning
- https://www.imagimob.com/

Table of contents

1 Overview

ModusToolbox™ is a set of tools to help you develop applications for Infineon devices. These tools include GUIs, command-line programs, software libraries, and third-party software that you can use in just about any combination you need. For more information, refer to the ModusToolbox™ tools package user guide. One part of the ModusToolbox™ ecosystem is the ML solution.

The ML 3.0 solution is a set of tools, libraries, and middleware that will help you build, evaluate and benchmark pre-trained ML models. The ML libraries easily and efficiently run the inference on an Infineon MCU. These libraries and tools help you rapidly deploy neural network (NN)-based regression and classification-type ML applications.

Note: Data collection and training algorithms are not part of the ModusToolbox™ ML 3.0 release.

The ML solution also provides a configurator to import pre-trained machine learning models and generate an embedded model (as C-code or binary file). This generated model can be used with the ML library along with your application code for a target device. The tool also lets you fit the pre-trained model of choice and evaluate its performance.

The ML models are often created on standard training frameworks such as TensorFlow, Keras, and PyTorch. They are kept in various formats such as H5, KERAS, TFLITE, ONNX, PTH, and PT2. The configurator converts the H5 or TFLite model to run in the Infineon MCU (any other format needs to be converted to one of the supported formats), as well as to run regression to assess performance degradation in the ML model, if any, in the process of conversion. If data is not available, it can load random data to do such verification. The most popular ML models are NN-based and the ModusToolbox™ ML tooling is targeted for NN-based ML models.

During the assessment, you can validate the performance of the optimized model by checking performance against test data (accuracy), and visualize the implementation resource requirements, such as the number of cycles to run the inference engine and the memory size of the NN model and inference working memory. The report includes data on the inference engine when running on a computer (reference model with the best accuracy and higher memory requirements), and on the target device (limited hardware resources).

For more information about the ML solution, visit this website:

https://www.infineon.com/cms/en/design-support/tools/sdk/modustoolbox-software/modustoolbox-machine-learning/

2 Getting started

To begin using the ML tools, you must first install the ModusToolbox™ tools package version 3.7.0 or higher. This tool is located on the Infineon website:

https://softwaretools.infineon.com/tools/com.ifx.tb.tool.modustoolboxpackmachinelearning

Alternatively, you can use the Infineon Developer Center (IDC) to install the ModusToolbox™ tools package and Machine Learning pack. Download the IDC launcher using this link:

https://www.infineon.com/cms/en/design-support/tools/utilities/infineon-developer-center-idc-launcher/

After installing the IDC tool, open the launcher and search for the following tools and install them:

- ModusToolbox tools package 3.7.0 or higher
- ModusToolbox Machine Learning pack 3.1.0 or higher

After all the tools are installed, you can get started with a code example. Infineon provides ML code examples, which include *README.md* files that guide you through the process to create and configure the application. There are a few ways to do this:

2.1 Creating a ModusToolbox™ application

- If you're new to the ModusToolbox™ environment, use the Eclipse IDE for ModusToolbox™. Refer to the quick start guide as needed.
- If you prefer not to use Eclipse, use the ModusToolbox™ Project Creator as a stand-alone tool and look for starter applications whose names start with "ML." Refer to the Project Creator user guide for more details.
- Currently, the ML solution supports PSOC™ 6 and PSOC™ Edge devices.

Note: *If you created an application before installing the Machine Learning pack, you must run* `make getlibs` *or use the Library Manager* ***Update*** *button to regenerate files before using the configurator for that application.*

2.2 Downloading a code example

The following are direct links to a couple of ML code examples:

https://github.com/Infineon/mtb-example-ml-profiler (PSOC™ 6)

https://github.com/Infineon/mtb-example-ml-gesture-classification (PSOC™ 6)

https://github.com/Infineon/mtb-example-ml-imagimob-mtbml-deploy (PSOC™ 6)

https://github.com/Infineon/mtb-example-psoc-edge-ml-profiler (PSOC™ Edge)

https://github.com/Infineon/mtb-example-psoc-edge-ml-deepcraft-deploy-audio (PSOC™ Edge)

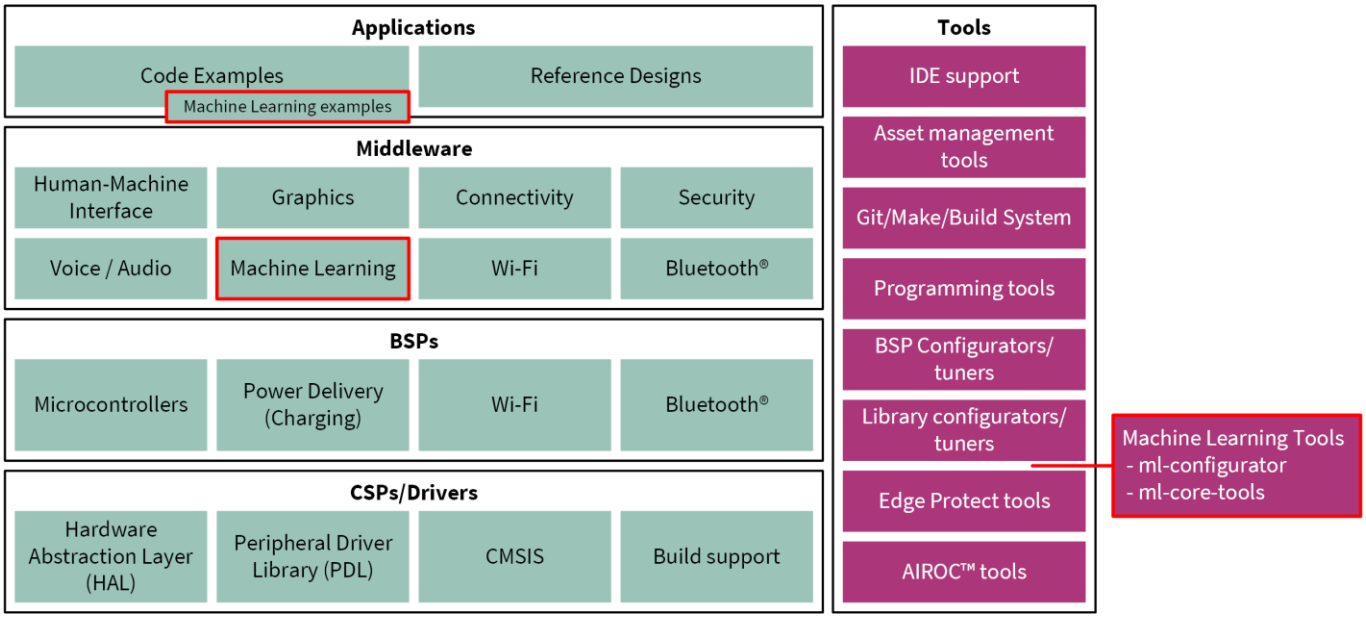
https://github.com/Infineon/mtb-example-psoc-edge-ml-deepcraft-deploy-motion (PSOC™ Edge)

As we add more code examples, you can access the complete list using the following link:

https://github.com/Infineon?q= -ml-%20NOT%20Deprecated

3 Features and architecture

The following diagram shows a very high-level view of what is available as part of ModusToolbox™ software. This is not a comprehensive list. It merely conveys the idea that there are multiple resources available to you. The ML solution is provided as a pack that you install in addition to the standard ModusToolbox™ tools package. The ML middleware is available through GitHub and used by the applicable code examples and reference designs.



3.1 Features

- Supports TFLite, Keras 2 (.h5), and Keras 3 (.h5, .keras) Model format
- Supports TensorFlow Lite for microcontrollers (TFLM) inference engine
- Supports the following characteristics of NNs:
  - Core NN Kernels: MLP, Conv1d, Conv2d, LSTM
  - Support NN Kernels: flatten, dropout, reshape, input layer
  - Activations: relu, softmax, sigmoid, linear, tanh
- Input Data Quantization Level:
  - 32-bit float
  - 16-bit integer
  - 8-bit integer
- NN Weights Quantization Level:
  - 32-bit float
  - 8-bit integer
- Supports multi-input and multi-output ML models
- Regression Data Evaluation
- Cycle and memory estimation
- PC based inference engine
- Target device-based inference engine (optimized)

Features and architecture

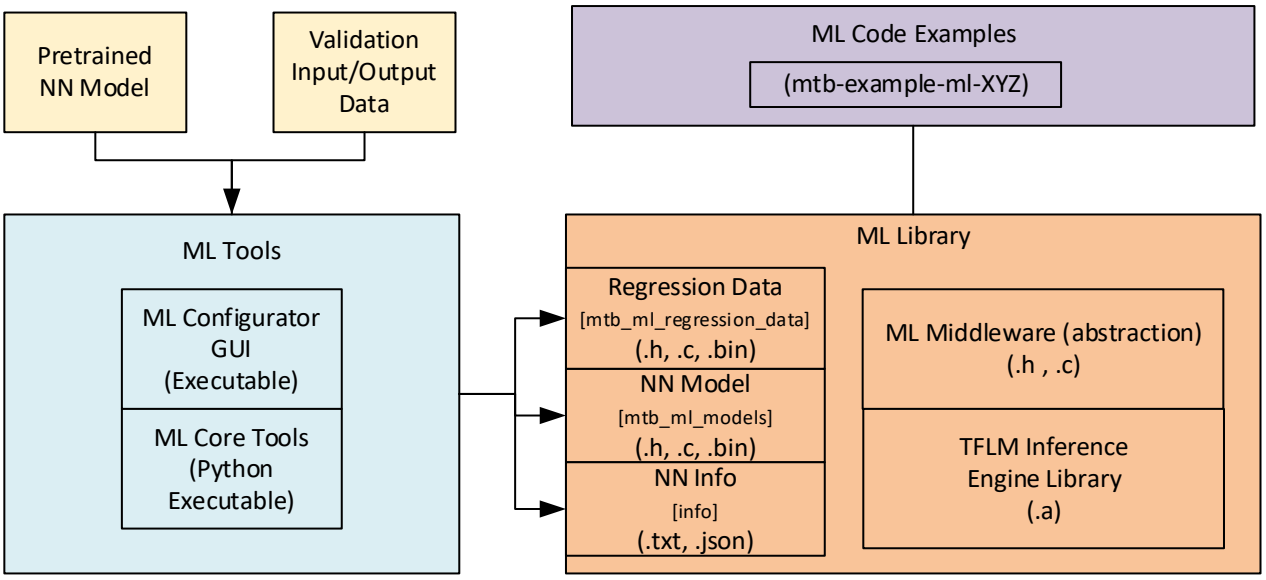
- Hardware acceleration with Ethos-U55 Neural Processing Unit (NPU)
- Hardware acceleration with NNLITE NPU

3.2 Components

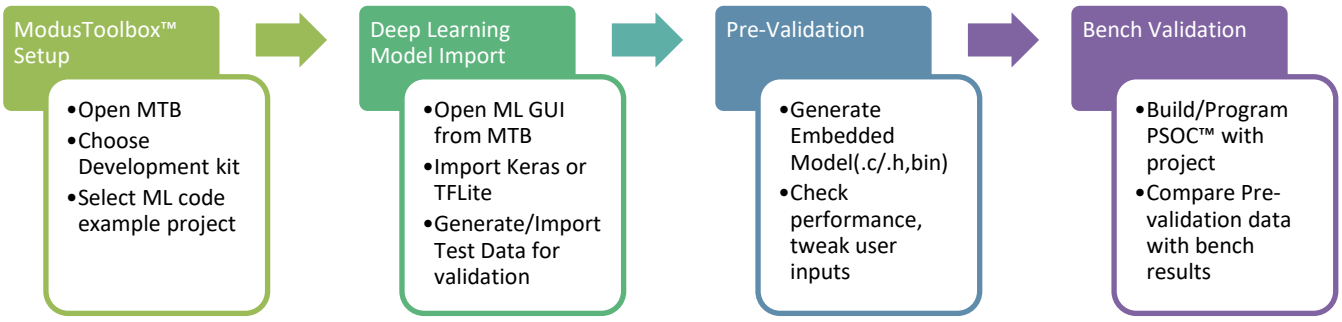
The ModusToolbox™ ML solution consists of four major components:

- ML Configurator: a GUI tool with options to load pre-trained models and test data. It also provides graphical plots and accuracy data when validating the optimized models against test data.
- ML core tool: a collection of Python scripts (provided as an executable) to perform parsing, regression and conversion of NNs.
- ML library: includes middleware helper functions, inference engine libraries to work with an NN model and input data.
- ML code examples: a collection of applications on how to use the ML libraries.

The following figure shows how these four assets interconnect with each other.



3.3 Development flow



3.4 Hardware requirements

This solution can run on any PSOC™ 6 MCU and PSOC™ Edge MCU. When it comes to designing the NN, pay special attention to the memory size and performance requirements to run the inference engine on the target device. The recommended hardware/kit platform for PSOC™ 6 MCU is the CY8CKIT-062S2-43012 or CY8CKIT-062S2-AI. Other PSOC™ 6-based platforms can be targeted as well, but please note the documentation and code examples may not work as expected. For PSOC™ Edge MCU, use the KIT-PSOCE84-EVK or KIT-PSOCE84-AI Evaluation Kits (EVK).

To better demonstrate the capability to run ML applications on the PSOC™ 6 MCU, sensor data needs to be sampled by the device. The CY8CKIT-028-SENSE IoT Sense Expansion kit provides a collection of sensors that are compatible with all Arduino-based PSOC™ 6 MCUs. For more details about this kit, visit this webpage: https://www.infineon.com/cms/en/product/evaluation-boards/cy8ckit-028-sense/.

If using one of the PSOC™ Edge EVKs, they already come with a variety of sensors to be used in ML applications.

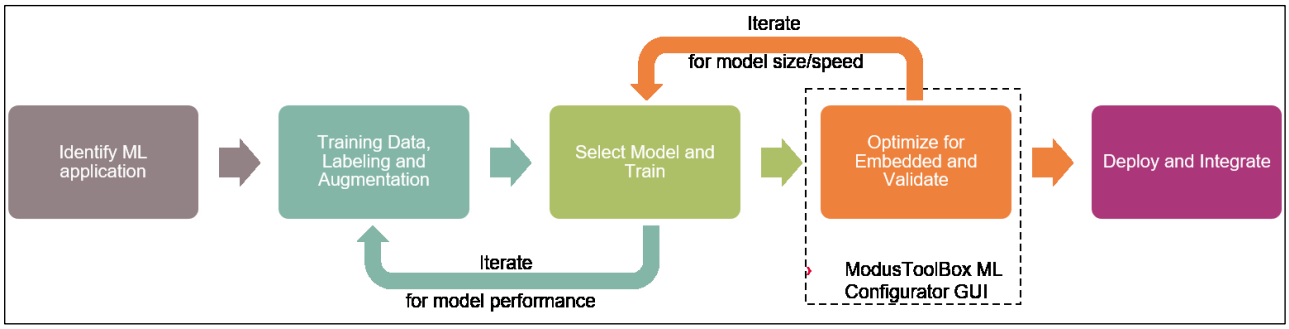
3.5 Software requirements

Refer to the Getting started section.

4 ModusToolbox™ ML Configurator

The ML Configurator accepts a pretrained ML model and generates an embedded model (as a C header or binary file), which can be used along with your application code for a target device. The tool lets you fit the pretrained model of choice to the target device with a set of optimization parameters. The tool is provided as a GUI and a command line tool that provide the same functionality. The GUI includes its own user guide, available from the **Help** menu. The command line tool includes a `-h` switch to display the various options available.

The following shows the design flow for a typical application. The ML Configurator forms a vital part in fitting the model to the target platform.



The ML Configurator supports only the TFLM inference engine. The next sections explain a few set of configurations to use with this inference engine.

Note: *The labeling, augmentation and training steps are handled by a third-party tool, such as DEEPCRAFT™ Studio, or machine learning framework, such as TensorFlow.*

Note: *When generating code or validating a model with the ML Configurator, you might see some warning messages related to the TensorFlow AutoGraph module. These warnings should not cause any problem and can be safely ignored. For more information about this, access the link provided in the warning message.*

4.1 Choosing a model

The first step when using the ML Configurator tool is to choose a pre-trained NN model.

Note: *There are many formats to define a NN model. The format of the NN depends on the platform used for training among other things.*

Note: *The TensorFlow framework supports Keras and TFLite model format. When generating Keras-H5 format with TensorFlow framework, you must use the TF v2.4.0 or later; otherwise, the model might fail when loading to the ML Configurator.*

The following table shows some popular formats:

Format	Supported in v3.x?	File extension	Link
Keras-H5	Yes	.h5, .hdf5	https://keras.io/
Keras 3	Yes	.keras, .h5	https://keras.io/
Tensorflow-protobuf	No	.pb	https://www.tensorflow.org/
TFLM	Yes	.tflite	https://www.tensorflow.org/lite

Format	Supported in v3.x?	File extension	Link
Caffe	No	.caffemodel .prototxt	https://caffe.berkeleyvision.org/
PyTorch	No	.pkl	https://pytorch.org/
Open NN Exchange	No	.onnx	https://onnx.ai/

If using the TFLM inference engine with a Keras model and quantization enabled, you must also provide calibration data. The following table shows coverage items by each supported format.

Format	Coverage
Keras (Layers)	Activation / Add / AveragePooling1D / AveragePooling2D / BatchNormalization / Clipped RELU / Concatenate / Conv1D / Conv2D / Dense / DepthwiseConv2D / Dropout / Flatten / GlobalAveragePolling1D / GlobalAveragePooling2D / GlobalMaxPooling1D / GlobalMaxPooling2D / InputLayer / LeakyReLU / LSTM / MaxPooling1D / MaxPooling2D / ReLU / Reshape / SeparableConv2D / Softmax / Transpose / Upsampling
TFLM (Operators)	Abs / Add / Add_N / Arg_Max / Arg_Min / Assign_Variable / Average_Pool_2D / Batch_MatMul / Batch_To_Space_Nd / Call_Once / Cast / Ceil / CircularBuffer / Concatenation / Conv_2D / Cos / CumSum / Depth_To_Space / Depthwise_Conv_2D / Dequantize / DetectionPostprocess / Elu / Equal / EthosU / Exp / Expand_Dims / Fill / Floor / Floor_Div / Floor_Mod / Fully_Connected / Gather / GatherNd / Greater / Greater_Equal / Hard_Swish / If / L2_Normalization / L2_Pool_2D / Leaky_Relu / Less / Less_Equal / Log / Log_Softmax / Logical_And / Logical_Not / Logical_Or / Logistic / Max_Pool_2D / Maximum / Mean / Minimum / MirrorPad / Mul / Neg / Not_Equal / Pack / Pad / PadV2 / Prelu / Quantize / Read_Variable / Reduce_Max / Reduce_Prod / Relu / Relu6 / Relu_0_to_1 / Relu_N1_to_1 / Reshape / Resize_Bilinear / Resize_Nearest_Neighbor / Round / Rsqrt / Shape / Sin / Slice / Softmax / Space_To_Batch_Nd / Space_To_Depth / Split / Split_V / Sqrt / Square / Squeeze / Strided_Slice / Squared_Difference / Sub / Sum / Svdf / Tanh / Transpose / Transpose_Conv / Unidirectional_Sequence_LSTM / Unpack / Var_Handle / While / ZerosLike

4.2 Importing test data

You can input your own test data for verification. Alternatively, you can use the ML Configurator to generate random test data for validation. To input the test data, click on the **Validate in Desktop** or **Validate on Target** tab and choose the desired settings. When providing the test data, it might include X (input) and Y (output) values, but the ML Configurator tool only uses the X data for validation. If the test data contains the Y, it can be label encoded or one-hot encoded (bitmap representation of the output).

The data can be stored with the following dataset structures:

4.2.1 In a single CSV file (ML format)

The CSV file must contain only numeric data with ‘,’ as field delimiter, with the exception of the header, which shall contain alphabet letters. The header is optional. All floating numbers must use ‘.’ For decimal points. The format shall be always UTF8 (no BOM).

You must specify the first feature column and the number of features in the ML Configurator. The tool supports 1D feature data only. If your model requires 2D or 3D feature data, use the NPZ format instead.

4.2.2 In a single NPZ format file (uses NumPy)

Features and labels can be loaded from a single NumPy compressed file. The feature and label must have been saved in the following order when creating the NumPy file:

```
import numpy as np
x = np.ones((n_samples, n_features))
```

```
y = np.ones((n_samples, n_labels))
np.savez('saved_features', x , y)
```

In this example, we populate x and y with "ones". In a real scenario, you should populate with actual data, where `n_samples` is the number of samples; `n_features` is the number of features; and `n_labels` is the number of labels. 2D and 3D feature data are also supported, represented as:

- 2D: (n_samples, a, b)
- 3D: (n_samples, a, b, c)

If a model requires multi-input or multi-output, the multiple inputs/outputs will be concatenated for a given sample.

4.2.3 In a folder format

This format separates the data by label. A base directory contains subfolders named by label, containing data files (nodes) with data for each label. The only supported node file format is JPEG.

4.3 Evaluating a model

Based on the imported test data, you can feed the input test data to the inference engine emulated on the computer or in the target, then analyze the result from the inference engine with the output test data. You may also choose the quantization in which the output model needs to be generated. There are three options:

- 8-bit
- 16-bit
- Floating number

The model conversion takes the reference model imported to the tool and converts it to C code for floating-point and integer. The floating-point mode uses input and model weights that are 32-bit precision and the regression on this converted model yields the highest accuracy. Although floating-point is more accurate than integer mode, it requires the most memory, and is not always desirable to embed on the target device.

In the ML Configurator, if you provide a Keras model, a model calibration group box appears. You can input some calibration data to improve the accuracy of the model after quantization. The format options of this calibration data are the same as the one used for validation. If you provide a **.tflite* model with the TFLM inference engine, there is no option to enable model quantization. However, if a **.tflite* model was created using int8x8 quantization, you must indicate the imported model has such quantization in the ML Configurator. By default, it assumes the **.tflite* model uses floating point.

The user also needs to select the model type:

- Classification: to identify discrete output variables based on the input variables
- Regression: to predict a continuous value based on the input variables

Another feature provided when using the TFLM inference engine is to run it without an interpreter (TFLM Interpreter-less), which can reduce the amount of memory required by the inference engine. However, this option doesn't allow run the validation in desktop option, only in the target. For more information about this feature, refer to the TFLM inference engine library section.

And last, the TFLM inference engine has also a feature to take advantage of sparsity if the trained model utilizes pruning. Note that this document does not cover any guidelines on how to prune a model during the training process. It assumes the developer provides a pre-trained pruned model, which then can be further optimized by packing sparse weights present in the model, saving memory when deploying the model to the target device. Also note that the sparsity feature has almost no effect on floating point models.

ModusToolbox™ ML Configurator

Once all the parameters are selected, the tool can provide an estimation in terms of cycles count to run the inference engine and the memory it requires (Model memory and Tensor Arena). Depending on the configuration chosen, some of the estimations are not available:

Device	Core	Memory estimation (on desktop)	Cycle estimation (on desktop)
PSOC™ 6	CM4	Yes	Yes
PSOC™ Edge	CM33+NNLITE	Yes	No
PSOC™ Edge	CM55+U55	Yes	U55 only

The cycle count estimator supports only a subset of all TFLM operators that have the most significant impact. The operators present in the model but not supported with the cycle estimator are reported as a warning. These are the following supported operators:

```
CONV_2D / REDUCE_MAX / DEPTHWISE_CONV_2D / TANH / MUL / ADD / AVERAGE_POOL_2D /
FULLY_CONNECTED / MAX_POOL_2D / MEAN / TRANSPOSE / CONCATENATION
```

If using an NPU, consider the support operators per NPU in the following table. The full list of NPU-supported operators on Ethos-U55 targets can be found at the official ARM support page. If the operator is not presented in any of the lists, it runs on CPU only.

Operator	NNLITE	Ethos-U55
ABS	Not supported	int8x8 / int16x8
ADD	int8x8 / int16x8	int8x8 / int16x8
MAXIMUM	Not supported	int8x8 / int16x8
MINIMUM	Not supported	int8x8 / int16x8
MUL	int8x8 / int16x8	int8x8 / int16x8
RSQRT	Not supported	int8x8
SQUARED_DIFFERENCE	Not supported	int8x8 / int16x8
SUB	int8x8 / int16x8	int8x8 / int16x8
TANH	int8x8 / int16x8	int8x8 / int16x8
LEAKY_RELU	int8x8 / int16x8	int8x8 / int16x8
LOGISTIC	int8x8 / int16x8	int8x8 / int16x8
RELU	int8x8 / int16x8	int8x8 / int16x8
RELU6 / RELU_0_TO_1 / RELU_N1_TO_1	int8x8 / int16x8	int8x8 / int16x8
PRELU	Not supported	int8x8 / int16x8
SOFTMAX	int8x8 / int16x8	int8x8 / int16x8
HARD_SWISH	Not supported	int16x8
CONV_2D	int8x8 / int16x8	int8x8 / int16x8
DEPTHWISE_CONV_2D	int8x8 / int16x8	int8x8 / int16x8
TRANSPOSE_CONV	Not supported	int8x8 / int16x8
AVERAGE_POOL_2D	int8x8 / int16x8	int8x8 / int16x8
MAX_POOL_2D	int8x8 / int16x8	int8x8 / int16x8
MEAN	Not supported	int8x8 / int16x8
FULLY_CONNECTED	int8x8 / int16x8	int8x8 / int16x8
UNIDIRECTIONAL_SEQUENCE_LSTM	int8x8	int8x8 / int16x8
RESIZE_BILINEAR	Not supported	int8x8 / int16x8
RESIZE_NEAREST_NEIGHBOR	Not supported	int8x8 / int16x8

Operator	NNLITE	Ethos-U55
CONCATENATION	Not supported	int8x8 / int16x8
EXPAND_DIMS	Not supported	int8x8 / int16x8
MIRROR_PAD	Not supported	int8x8 / int16x8
PAD	Not supported	int8x8 / int16x8
QUANTIZE	Not supported	int8x8 / int16x8
RESHAPE	Not supported	int8x8 / int16x8
SLICE	Not supported	int8x8 / int16x8
SPLIT	Not supported	int8x8 / int16x8
SPLIT_V	Not supported	int8x8 / int16x8
SQUEEZE	Not supported	int8x8 / int16x8
STRIDED_SLICE	Not supported	int8x8 / int16x8
TRANSPOSE	Not supported	int8x8 / int16x8
UNPACK	Not supported	int8x8 / int16x8
ARG_MAX	Not supported	int8x8 / int16x8

When using Ethos-U55, you can find more information about the number of operators running on Ethos-U55 in the following folder: *mtb_ml_gen/info/*. Refer to the Exporting model on target device section for more information.

When using NNLITE, you can review the log generated by the ML-configurator tool. It lists all the operators running on NNLITE.

4.3.1 Vela compiler options

If using the CM55+U55 core option, the ML Configurator provides additional options related to the vela compiler, which is a software tool that compiles a TFLM model into an optimized version that runs on Ethos-U55 NPU. The following table explains each option:

Parameter name	Options	Description
System config	PS84_M55_U55_400MHz	The only option available assumes the CM55 and U55 run at 400 MHz
Memory mode	Sram_Only	Default option. Weights and tensor arena are placed in SOCMEM
	Shared_Sram	Weights are placed in the external memory and tensor area is placed in SOCMEM
Optimize	Performance	Default option. Optimizes the model for best performance
	Size	Optimizes the model for minimum size
Tensor allocator	HillClimb	Default option. Heuristic-based approach that aims to minimize memory fragmentation and reduce peak memory usage during model execution
	Greedy	Allocates tensors based on their size and availability of free memory
	LinearAlloc	Allocates tensors one after another in a contiguous block of memory
Arena cache size (Bytes)	<Integer number>	Set the size of the arena cache memory area, in bytes.
Recursion limit	<Integer number>	Default is 1000. Set the recursion depth limit.
Max block dependency	1, 2 or 3	Default is 3. Set the maximum value that can be used for the block dependency between NPU kernel operations.

The .tflite models that have already been optimized with Vela can be processed by the Configurator. Models are automatically recognized by the ML Configurator as Vela-optimized, so no user-input is needed.

4.4 Validating a model

The ML Configurator provides two options to validate the model:

- Validation in Desktop
- Validation on Target

When validating the device, a report tells if the model passed or failed. It considers as a pass when the accuracy is equal or greater than 98%. As mention previously, it also reports the amount of RAM memory required to run the model. Note the following limits when running on target and desktop:

Device	Core	On target (internal memory)	On desktop (for emulation purposes)
PSOC™ 6	CM4	1.0Mb (SRAM)	4.0Mb
PSOC™ Edge	CM33	0.5Mb (SRAM)	16.0Mb
PSOC™ Edge	CM55	5.0Mb (SocMEM)	32.0Mb

If the memory on target device is not large enough, you might need to run some external RAM memory to inference the model.

When validating on target, the regression data provided by the user is streamed over the UART, then the response is return and compared to the reference. The tool calculates the final accuracy and the errors between the returned output and the reference. Refer to section 4.6 for more details.

For accurate cycle count measurements, you can run the model in the target device to acquire some profiling information. Refer to the mtb-example-ml-profiler (PSOC™ 6) and mtb-example-psoc-edge-ml-profiler (PSOC™ Edge) for more details.

Note: *Note Validation in Desktop is not available when using the TFLM interpreter-less option or selecting a NPU - (CM55 + U55 or CM33 + NNLITE).*

4.5 Exporting model on target device

When the model has been converted with an acceptable accuracy and satisfied the memory requirements for the target device, it can now be exported on the target device. The ML Configurator generates a header file or binary file containing the simplified model with the desired quantization. It can also export regression data to profile the model in the target device.

All this data can be exported to a directory you specify, which usually is part of the application in development. The following shows a summary of files exported by the tool. These files depend on the inference engine and quantization selected.

Assume the name of the model is "NN" and the root folder is "mtb_ml_gen". The "<type>" can assume *float*, *int8x8 or int16x8*, and the "<inference>" can assume *tflm* (runtime interpreter) or *tflm_less* (interpreter-less).

Folder / File Name	Description
/mtb_ml_gen/	Root folder. Name is chosen by the user
/mtb_ml_gen/info/	General information folder about the model
NN_vela_<type>.txt	(CM55 only) Contains report from Vela compiler
/mtb_ml_getn/info_target	Contains the report when validating the model on target
device_log.txt	Device log from UART when communication with the target
profiler_info.txt	Profiling info of the model when using the streaming.
/mtb_ml_gen/model_gen_dir/	Folder that stores modified/converted model files (for internal use only)

ModusToolbox™ ML Configurator

Folder / File Name	Description
/mtb_ml_gen/mtb_ml_models/	Folder that contains the model weights/bias and parameters
NN_<inference>_model_<type>.h	Header file with model array [tflm] or function [tflm_less] declaration
NN_<inference>_model_<type>.c	C file containing the flat-buffer array definition for weights/bias [tflm]
NN_<inference>_model_<type>.cpp	C++ file containing the implementation of TFLM functions [tflm_less]
NN_<inference>_model_<type>.bin	Weights/bias array in binary format [tflm]
/mtb_ml_gen/mtb_ml_regression_data/	Folder that contains regression data
NN_<inference>_x_data_<type>.h	Header file with input data array declaration
NN_<inference>_x_data_<type>.c	C file containing the input data array definition
NN_<inference>_x_data_<type>.bin	Input data array in binary format
NN_<inference>_y_data_<type>.h	Header file with output data array declaration
NN_<inference>_y_data_<type>.c	C file containing the output data array definition
NN_<inference>_y_data_<type>.bin	Output dt array in binary format

Note: *Multiple models can be stored in the same folder, since the name of the MODEL is used as a prefix for all the files.*

Note: *Regression data generated for the "tflm" and "tflm_less" is the same, so the <inference> is set to "tflm" for both.*

The binary files are useful when storing the data in a file system in the target device. If using header files, the data is available as C arrays with the name format shown in the following table. The header file also provides some `#define` about the model:

File name: *NN_<inference>_model_<type.h>*	
Array name	**Description**
`NN_model_bin[]`	Weights/bias
DEFINE name	**Description**
`NN_MODEL_BIN_LEN`	Length of the weights/bias array in bytes
`NN_ARENA_SIZE`	Size of the TFLM arena in bytes

File name: *NN_<inference>_x/y_data_<type>.h*	
Array name	**Description**
`NN_x_data_bin[]`	Input data
`NN_y_data_bin[]`	Output data
DEFINE name	**Description**
`NN_X_DATA_BIN_LEN`	Length of the input data in bytes
`NN_Y_DATA_BIN_LEN`	Length of the output data in bytes

Note: *The NN_<inference>_model_<type.h> file contains an array with the model weights. By default, it stores it in the flash and loads automatically to the RAM. There is an option to place this array in a specific section of the memory. Simply define the `CY_ML_MODEL_MEM` macro in the Makefile to a section available in the linker script. In terms of performance, the inference engine runs faster when placing the model weights in RAM, and slower if placing in the external memory.*

ModusToolbox™ ML Configurator

When using PSOC™ 6

To place to the internal flash only, you can add:

```
DEFINES+=CY_ML_MODEL_MEM=.constdata
```

To place to external memory, you can add:

```
DEFINES+=CY_ML_MODEL_MEM=.cy_xip
```

Refer to the https://github.com/Infineon/mtb-example-PSOC6-qspi-xip code example to properly setup the XIP mode using QSPI.

This approach does not work with the PSOC™ 64 family, due to the protection settings to work in XIP mode.

When using PSOC™ Edge

To place to the external flash, you can add:

```
DEFINES+=CY_ML_MODEL_MEM=.constdata
```

To place to the internal SOCMEM, you can add (best performance for CM55 + U55):

```
DEFINES+=CY_ML_MODEL_MEM=.cy_socmem_data
```

To place to the internal SRAM0 bank, you can add (best performance for CM33 + NNLITE):

```
DEFINES+=CY_ML_MODEL_MEM=.cy_sram_code
```

The following table shows the main differences between using C header files and binary files.

File Format	Requires File System	Easy to Upgrade	Access	Scalability
Header File	Optional	No. Part of the firmware image	Easy. Directly access a variable in firmware	Low. Usually placed on internal memory.
Binary	Recommended	Yes. Update a file in the filesystem	Hard. Need to open and load a file	High. Usually placed on external memory

When using the TFLM Interpreter-less feature, the model file generated contains functions instead of arrays. The following table shows a summary of the functions and defines.

File name: *NN_tflm_less_model_<type.h>*

Function name	Description
`NN_init()`	Sets up the model with init and prepare steps
`NN_input(index)`	Returns the input tensor with the given index
`NN_output(index)`	Returns the output tensor with the given index
`NN_invoke()`	Runs inference for the model
`NN_inputs()`	Returns the number of input tensors
`NN_outputs()`	Returns the number of output tensors
`NN_input_ptr(index)`	Return the buffer pointer of input tensor
`NN_input_size(index)`	Return the buffer size of input tensor
`NN_input_dims_len(index)`	Return the dimension size of input tensor
`NN_input_dims(index)`	Return the dimension buffer pointer of input tensor
`NN_output_ptr(index)`	Return the buffer point of output tensor
`NN_output_size(index)`	Return the buffer size of the output tensor
`NN_output_dims_len(index)`	Return the dimension size of output tensor

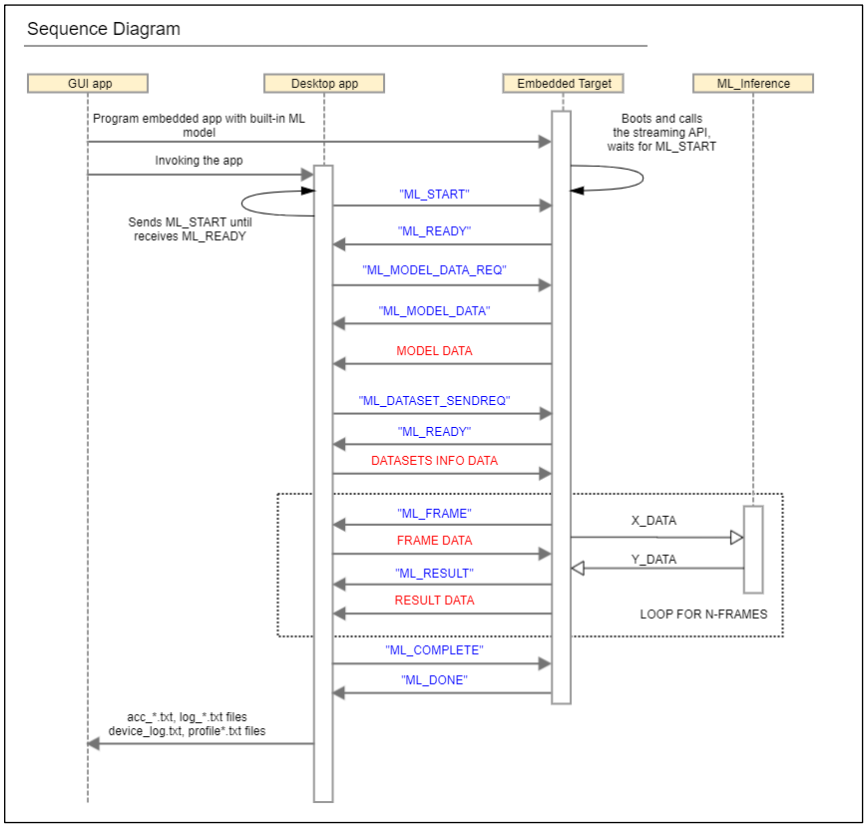
File name: *NN_tflm_less_model_<type.h>*	
`NN_output_dims(index)`	Return the dimension buffer pointer of output sensor
DEFINE name	**Description**
NN_MODEL_CONST_DATA_SIZE	Non-volatile data size used by this model
NN_MODEL_INIT_DATA_SIZE	Initialized volatile data size used by this model
NN_MODEL_UNINIT_DATA_SIZE	Uninitialized volatile data size used by this model

The input regression data (*x_data*) contains not only the input data, but also the data input type, number of inputs in the first layer of the NN, the number of samples and the Q-format. Here is how to parse the data:

Location	**Description**
byte 0	Data input type: 0 → Unknown 1 → Float 32-bit 2 → Integer 8-bit 3 → Integer 16-bit
byte 1	
byte 2	
byte 3	
byte 4	Number of samples in the dataset
byte 5	
byte 6	
byte 7	
byte 8	Number of nodes in the first layer (input) or frame size of the sample dataset
byte 9	
byte 10	
byte 11	
byte 12	If using streaming RNN model, the number of the time steps. If using non-streaming RNN model, this value equal one (1). If using any other model, this value equal minus one (-1).
byte 13	
byte 14	
byte 15	
byte 16	Data samples
byte …	

4.6 Streaming regression data to the Device

The ML 3.0 solution comes with a feature that enables streaming the regression data to the target device through the UART. Instead of storing the regression data locally, as explained in the previous section, the ML Configurator tool streams the exact same data to the target device. Here is the overall sequence diagram of the protocol in place.



The simplest way to leverage the streaming capability is to use the mtb-example-ml-profiler code example. Ensure the baud rate configured in the ML configurator tool is the same as configured in the code example.

To change the baud rate in the code example, open the *main.c* file and set the correct speed to the `cy_retarget_io_init()` function.

4.7 Using command line

The ML Configurator comes with a command line executable, named as *ml-configurator-cli*. To see all the options available, you can type:

```
ml-configurator-cli -h
```

This command prints the following message:

```
Options:
  -?, -h, --help              Displays help on command line options.
  --help-all                  Displays help including Qt specific options.
  -v, --version               Displays version information.
  -c, --config <config_file>  Path to the
                              configuration file.
  -o, --output-dir <dir>      The path to the
                              generated source directory. It is either an
                              absolute path or a path relative to the
                              configuration file parent directory.
  --convert                   Analyze and convert
  --evaluate                  Evaluate
```

ModusToolbox™ ML Configurator

The option `--config` requires a configuration file with extension MTBML, which is generated by the ML Configurator GUI.

If the configurator is not used to generate a *.mtbml* file, then one must be generated before using the ml-configurator-cli tool. The following example should be used as a template. The JSON file must have an extension of *.mtbml* and use UTF-8 encoding.

Formatted JSON example:

```
{
    "app": "ML",
    "calibration_data": {
        "active_state": true,
        "feat_col_count": 784,
        "feat_col_first": 1,
        "input_calibration_type": "ML",
        "input_format": "JPEG",
        "path": "test_data/test_data.csv",
        "target_col_count": 1,
        "target_col_first": 0
    },
    "filetype": "modustoolbox-ml-configurator",
    "formatVersion": "3",
    "lastSavedWith": "ML Configurator",
    "lastSavedWithVersion": "3.0.0",
    "model": {
        "model_task": "CLASSIFICATION",
        "optimization_tflm": false,
        "path": "pretrained_models/small_mlp_mnist.h5",
        "quantization": {
            "float32": true,
            "int16x8": true,
            "int8x8": true
        },
        "sparsity_tflm": false,
        "tflm_model_quantization": "FLOAT",
        "verbose_output": false
    },
    "name": "TEST_MODEL",
    "output_dir": "mtb_ml_gen",
    "target": "PSOC6",
    "toolsPackage": "ModusToolbox Machine Learning Pack 3.0.0"
    "validation": {
        "feat_col_count": 784,
        "feat_col_first": 1,
        "input_format": "JPEG",
        "input_type": "ML",
        "max_samples": 1000,
        "path": "test_data/test_data.csv",
        "quantization": {
            "float32": true,
            "int16x8": true,
            "int8x8": true
        },
        "target": {
            "target_baud_rate”: 1000000,
            "target_quantization": "int8x8"
        }
    }
}
```

ModusToolbox™ ML Configurator

The JSON breakdown table gives a description of each object and can be used to edit the example JSON file. All fields need to be included, even if not used.

Objects	Description	Value
calibration_data		
active_state	Enables or disables model calibration section	true false
feat_col_count	Number of feature columns	Integer
feat_col_first	Index of first feature column	Integer
input_calibration_type	Input data type	NPZ FOLDER ML
input_format	Only used if input_calibration_type=ML. Format type for the input data.	JPEG CSV
path	Path to calibration data	String path
target_col_count	Number of target columns	Integer
target_col_first	Index of first target column	Integer
Filetype		
filetype	Identifies the file type	modustoolbox-ml-configurator
Model		
model_task	Type of the model	CLASSIFICATION REGRESSION
optimization_tflm	Enables or disables the "tflm_interpreter_less" setting for "tflm" inference engine. • false (default) – disables "tflm interpreter less" setting, • true – enables "tflm interpreter less" setting.	true false
path	Path to model file. If the path is relative, it is evaluated with respect to the location of the configuration file.	String path
sparsity_tflm	Enable use of a memory-efficient packed format for any sparse weights present in the model	true false
tflm_model_ quantization	Define if the tflite model was created using fixed-point or floating point.	FLOAT INT8X8 INT16X8
verbose_output	Print more debug information	true false
model.quantization		
float32	Generate source for quantization type	true false
int8x8	Generate source for quantization type	
Int16x8	Generate source for quantization type	
name		
name	Project name (used as output file prefix)	User defined
output_dir		
output_dir	Output directory. If the path is relative, it is evaluated with respect to the location of the configuration file.	String path
target		
target	Specifies the device	PSOC6 PSE84_M33_NNLITE2 PSE84_M55_U55

ModusToolbox™ ML Configurator

Objects	Description	Value
validation		
feat_col_count	Number of feature columns	Integer
feat_col_first	Index of first feature column (0-based)	Integer
input_format	Format type for the input data	CSV BIN JPEG
input_type	Input data type, can be PRNG, NPZ, FOLDER, or ML	PRNG NPZ FOLDER ML
max_samples	Number of random samples to generate	Integer [0~1000]
path	Path to validation input data file or folder. If the path is relative, it is evaluated with respect to the location of the configuration file.	Path
validation.quantization		
float32	Validate quantized model type (true or false)	true false
Int16x8	Validate quantized model type (true or false)	
int8x8	Validate quantized model type (true or false)	
validation.target		
target_quantization	Which quantization type to use for "Evaluate on Target"	int8x8 int16x8 float32
target_col_count	Number of feature columns	Integer
target_col_first	Index of first target column (0-based).	Integer
app		
app	Application type. Currently, it only accepts “ML”.	ML
formatVersion		
formatVersion	File format version. The version described here is 3. The tool should refuse to load files with a higher version that the tool’s maximum supported version. The file format version is incremented when a breaking change is made to the format. Adding an optional field is not a breaking change. Changing the type of an existing field is a breaking change.	3
lastSavedWith		
lastSavedWidth	Tool that was last saved with. Currently, it only accepts “ML Configurator”	ML Configurator
lastSavedWithVersion		
lastSavedWidthVersion	Tool version that was last saved with. The version described here is 3.0.0. The tool version is incremented when a new release is publicly available.	3.0.0
toolsPackage		
toolsPackage	Name of the package.	ModusToolbox Machine Learning Pack 3.0.0 / ModusToolbox PSOC E84 Early Access Pack

5 ModusToolbox™ ML embedded libraries

For the ML 3.0 solution, there are two middleware assets deployed as ModusToolbox™ libraries:

- TFLM inference engine
- ML middleware

5.1 TFLM inference engine library

The TFLM library is runs machine learning models on Infineon microcontrollers. The TFLM library is available as a ModusToolbox™ asset. Use the following GitHub link:

https://github.com/infineon/ml-tflite-micro

You can add a dependency file (mtb format) under the *deps* folder or use the Library Manager to add it in your project. It is available under **Library > Machine Learning > ml-tflite-micro**.

This document does not provide details on how to use the TFLM APIs written in C++. It is recommended to use our machine learning abstraction middleware, explained in the ML middleware library section. For more general information about the TFLM, including examples and documentation, refer to this link:

https://www.tensorflow.org/lite/microcontrollers

Infineon provides the following methods to deploy machine learning models using the TFLM inference engine:

- **TFLM runtime interpreter**: Uses an interpreter to process a machine learning model deployed as binary data. This allows easy updates on the deployed model or the need to inference multiple models in the application.
- **TFLM interpreter-less**: Does not require a run-time interpreter; instead uses pre-generated code to execute the inference. This allows smaller binary and less overhead on inference execution.

In both cases, this version of the library provides two types of quantization – *floating point* and *8-bit integer*. To use this library, the following `COMPONENTS` and `DEFINES` are required:

- If using TFLM runtime interpreter:

  ```
  DEFINES+=TF_LITE_STATIC_MEMORY
  COMPONENTS+=ML_TFLM
  ```

- If using TFLM interpreter-less (note that Ethos-U55 does not support this option):

  ```
  DEFINES+=TF_LITE_STATIC_MEMORY TF_LITE_MICRO_USE_OFFLINE_OP_USER_DATA
  COMPONENTS+=ML_TFLM_LESS
  ```

- If using floating point:

  ```
  COMPONENTS+=ML_FLOAT32
  ```

- If using 16-bit integer:

  ```
  COMPONENTS+=ML_INT16x8
  ```

- If using 8-bit integer:

  ```
  COMPONENTS+=ML_INT8x8
  ```

- If using CPU Only, such as PSOC™ 6 ARM Cortex CM4:

  ```
  COMPONENTS+=ML_CPU_ONLY
  ```

- If using NNLITE:

  ```
  COMPONENTS+=NNLITE2
  ```

- If using the Ethos-U55:

```
COMPONENTS+=U55
```

Note: *Defining the COMPONENTS ML_FLOAT32/ML_INT16x8/ML_INT8x8 is optional. If none of these components are defined, type-less variable of middleware API will be used, identifying quantization runtime.*

5.2 ML middleware library

This library works as an abstraction layer to the TFLM inference engine library. It implements all the steps described in the previous sections in the following functions:

```
mtb_ml_model_init()    : to initialize the model (call once)

mtb_ml_model_get_output_tensor() : to obtain the output buffer pointer and its
size (call once on every output)

mtb_ml_model_inputs() : to feed data for each input (call on every sample)

mtb_ml_model_invoke() : to run the inference engine (call on every sample)
```

Note: *The **mtb_ml_model_run()** was replaced with **mtb_ml_model_invoke()**. The mtb_ml_model_run() can still be used for single-input and single-output models. Multi-input and multi-output models must use the new mtb_ml_model_invoke() function in combination with the mtb_ml_model_inputs() function.*

Note: *For multi-input and multi-output models, you can extract the number of inputs and outputs from the model object structure variables input_count and output_count, respectively.*

The library also provides helper MACROs to include the model files generated by the ML Configurator tool. For example, if a model was generated using the output file prefix "test_model", you can use:

```
#include MTB_ML_INCLUDE_MODEL_FILE(test_model)            // Model file
#include MTB_ML_INCLUDE_MODEL_X_DATA_FILE(test_model)     // Regression data X
#include MTB_ML_INCLUDE_MODEX_Y_DATA_FILE(test_model)     // Regression data Y
```

Note: *The includes above are only valid when including one of the ML_FLOAT32/ML_INT16x8/ML_INT8x8 COMPONENTS. If they are not included, you need to manually write the file's name.*

The `mtb_ml_model_init()` function has an option to skip the parsing and memory allocation by providing pointers for the scratch and persistent memories (for Infineon inference engine only). The application can use the following helper MACRO to know how much memory to allocate:

```
// TFLM inference engine
MTB_ML_MODEL_ARENA_SIZE(test_model)
```

To access the data arrays generated for the model, use the following MACROs:

```
MTB_ML_MODEL_NAME_STR(test_model)     // String for the model name
MTB_ML_MODEL_BIN_DATA(test_model)     // Populates the model binary structure
MTB_ML_MODEL_X_DATA_BIN(test_model)   // Regression data input array
MTB_ML_MODEL_Y_DATA_BIN(test_model)   // Regression data output array
```

The following table shows the steps for the two initialization methods:

ModusToolbox™ ML embedded libraries

Method 1: Using internal memory allocation

```
mtb_ml_model_t *model_object;

/* NN model data */
mtb_ml_model_bin_t model_bin = {MTB_ML_MODEL_BIN_DATA(test_model)};

/* Initialize the model */
mtb_ml_model_init(&model_bin, NULL, &model_object);
```

Method 2: Using external memory allocation

```
mtb_ml_model_t *model_object;

uint8_t tensor_arena[MTB_ML_MODEL_ARENA_SIZE(test_model)];
mtb_ml_model_buffer_t mem_buf = {tensor_arena,
                                 MTB_ML_MODEL_ARENA_SIZE(test_model)};

/* NN model data */
mtb_ml_model_bin_t model_bin = {MTB_ML_MODEL_BIN_DATA(test_model)};

/* Initialize the model */
mtb_ml_model_init(&model_bin, &mem_buf, &model_object);
```

Note: *When using `ML_TFLM_LESS`, only method 1 applies.*

Note: *When using Ethos-U55, the tensor arena must be placed in SOCMEM. Method 1 automatically uses SOCMEM if the heap is placed in SOCMEM (default option). Method 2 requires explicit placement if declared statically.*

Regardless of the initialization method, it is important to note that `mtb_ml_model_init()` should not be called more than once, as unpredictable behavior may occur.

This library also comes with a set of APIs to handle streaming data from the ML Configurator tool. It handles the UART connection and what profiling and debugging features to enable. The sequence of functions to call are:

```
mtb_ml_stream_init() : execute the interface initialization

mtb_ml_model_get_output_tensor() : get output buffer pointer and its size

LOOP:

   mtb_ml_stream_input_data() : get data from the host interface

   mtb_ml_model_inputs() : add data for each input

   mtb_ml_model_invoke() : run the inference engine

   mtb_ml_stream_output_data() : send data to the host interface

mtb_ml_inform_host_done() : tell the host task is complete
```

When using RNN models, additional steps are required:

- Reset the internal states on every frame by calling - `mtb_ml_model_rnn_reset_all_parameters()`
- Create “slices” of 2D array data from 1D array generated by the ML-configurator as input for the `mtb_ml_model_inputs()`

Note: *Stateful RNN is not supported in TFLM_LESS mode.*

ModusToolbox™ ML embedded libraries

The following link provides an example using the streaming feature of this solution:

https://github.com/infineon/mtb-example-ml-profiler

https://github.com/infineon/mtb-example-psoc-edge-ml-profiler

This library also comes with helper functions that:

- convert floating-point to fixed-point and vice versa
- quantize the inputs
- dequantize outputs
- return the index of the maximum value in an array

If using Arm Cortex CM55 with Helium extension, add the following:

```
DEFINES+=ARM_MATH_HELIUM ARM_MATH_DSP
```

The solution implements its own version of the CMSIS_NN library, therefore do NOT add CMSIS_NN in the component list in the Makefile, only the header files, as the example below:

```
INCLUDES+=$(SEARCH_cmsis)/COMPONENT_CMSIS_NN/Include
```

When using Ethos-U55 with CM55, the underneath TFLM implementation, clear/invalidate the CM55 D-Cache to avoid any discrepancy in the memory (as the latest data might be only available in the cache or actual memory). This is done for every TFLM operator. However, if all operators are handled by the Ethos-U55, there is no need to clear/invalidate the cache on every operator, only in the input and output layer. You can add the following macro in the CM55 Makefile DEFINES list to disable these cache operations:

```
DEFINES+= MTB_ML_ETHOSU_CACHE_MGMT_TYPE=MTB_ML_ETHOSU_CACHE_MGMT_OUTER_LAYERS
```

If you see any changes in the inferencing results, that means you should NOT enable this cache optimization, as the CM55 and U55 are accessing the same internal buffers.

You can also call `mtb_ml_set_cache_mgmt_type()` function to change the cache management type to one of the below options dynamically.

- **MTB_ML_ETHOSU_CACHE_MGMT_CONDITIONAL**: this mode clears or invalidates the entire cache based on an internal state. It can reduce the total number of CPU cycles, but it might cause some undesired behavior in the application.
- **MTB_ML_ETHOSU_CACHE_MGMT_ALL_LAYERS**: this mode clears and invalidates the cache by address for each layer using cache-API calls within the driver (default option).
- **MTB_ML_ETHOSU_CACHE_MGMT_OUTER_LAYERS**: this mode clears the input layer before executing the inference and invalidates the output layer after executing inference. Should be only used if all operators are supported by Ethos-U55. It provides always the best performance.

The `mtb_ml_set_cache_mgmt_type()` function should be used when you have multiple models to run sequentially in your application, and they require different cache management types.

6 ModusToolbox™ ML core tools

The ML core tools are the backend utilities which the ML Configurator tool interacts with to perform a variety of features:

- Parsing a JSON file for details on the model, data and configuration
- Creating random data for regression
- Alternative to using random data, imported data can be created [2]
- Loading pretrained Keras 2 (.h5), Keras 3 (.h5, .keras), or TFLITE model and parsing them for ModusToolbox™ ML relevant information
- Performing model reference evaluation on the specified dataset to determine a base floating-point collection for reference metrics
- Converting the pretrained or trained model to a library C code for embedding and use on the PSOC™ 6 or PSOC™ Edge MCU; and as part of the ModusToolbox™ ecosystem
- Saving log files for output and debugging purposes during and after their operation
- Regression data generation, tests and cross-domain verification
- Stream regression data to the target device

This tool is written using Python scripts and released as an executable, which is available with the Machine Learning pack. It is meant to be used with the ML Configurator only.

Note: *The Machine Learning Tech Pack 3.0.0 contains a ML core tools bundle that is intended for ARM-based Mac computers (M1/M2/M3). To make the pack work on Intel-based Mac machines, there is a dedicated ML core tools ZIP file. Download this from the Infineon Developer Center. Replace the contents of the ML coretools directory (<path_to_pack>\tools\ml-coretools) with the unzipped file.*

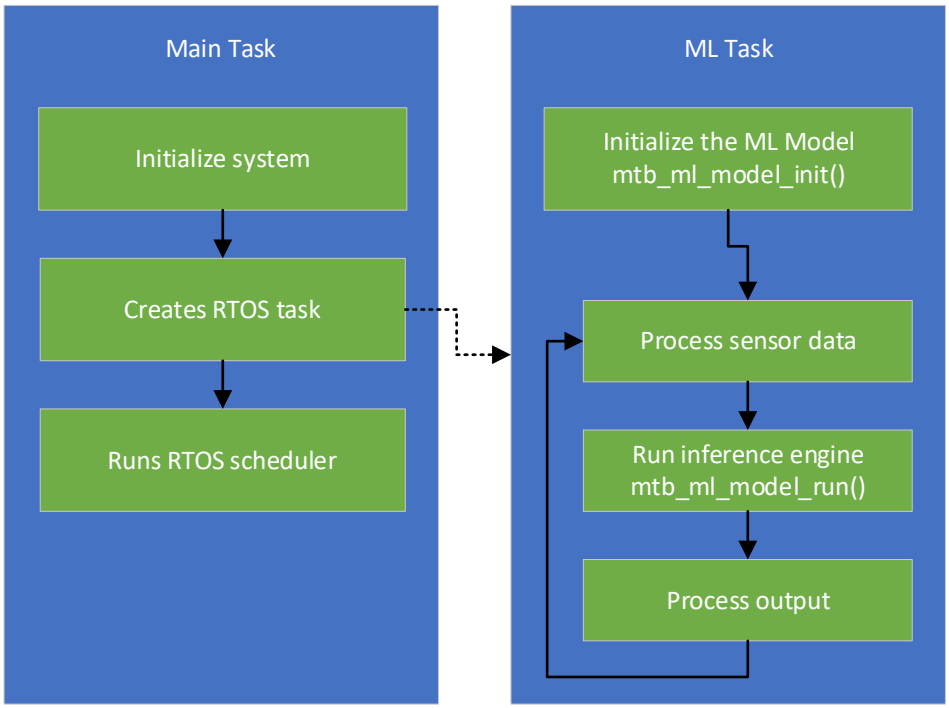
[2] This reads data from files structure in one of the ModusToolbox™ ML formats.

7 ModusToolbox™ ML integration

The easiest way to run the ML solution with PSOC™ 6 and PSOC™ Edge MCUs is by leveraging the code available in the ML code examples.

7.1 Using an RTOS

If using an RTOS, the most effective way to integrate the inference engine in your project is to use the ML middleware library to wrap all interactions with the TensorFlow Inference Engine library. You can have a single task or multiple tasks to handle different functions of the application. The following example shows a task handling all related machine learning functions.



As the RTOS typically handles memory management, you need to ensure that the RTOS heap has enough memory for the scratch and persistent memory required by the inference engine, which can be quite large depending on the model being executed.

7.2 Handling inputs and outputs

The ML inference engine can use both floating-point and fixed-point inputs. One good practice before feeding a NN is to normalize the input. There are different types of normalization in statistics. In this section, we refer to the min-max feature scaling type. If the input data is not normalized in the embedded firmware due to limited hardware resources, the training framework tool also cannot normalize the input data. The input data fed to the inference engine and the training algorithm need to match.

If using floating-point as input, the normalization is straight forward. It depends on the maximum and minimum value of the sensor, and the normalization range. Usually the normalization is between [0,1] or [-1,1]. Use the following formula to normalize:

$$NN\ input\ data = N_{MAX} - (S_{MAX} - Sensor\ Data) \times \frac{N_{MAX} - N_{MIN}}{S_{MAX} - S_{MIN}}$$

Where S_{MAX} and S_{MIN} are the maximum and minimum value the sensor can produce, and N_{MAX} and N_{MIN} are the normalization range. If the sensor data needs to be filtered, or subjected to some sort of processing (besides normalization), it must mimic the steps of the training.

7.2.1 Using TFLM Inference Engine

The TFLM inference engine uses asymmetric quantization, which uses a zero-point and scaler variables. The following equation applies:

$$FloatValue = (IntegerValue - ZeroPoint) * Scaler$$

The zero-point and scaler values are defined during the model calibration, which requires the user to provide some calibration data when generating the model files. The input and output of a TFLM model might have different values for the zero-point and scaler.

To quantize the results from float to integer, you can use the `mtb_ml_utils_model_quantize()` function, which converts a given input float array to integer using the input zero-point and scaler from the model. This information is stored internally in the `mtb_ml_model_t` structure.

To de-quantize the results to some meaningful value, you can use the `mtb_ml_utils_model_dequantize()` function, which provides the float representation of the model's output by using the output zero-point and scaler from the model.

For TFLM inference using int8x8 quantization, the model calibration data plays an important role on the accuracy of the model. If the data provided is not representative enough to stimulate the network to get good estimates of activation value statistics (min/max value range), it can result in poor network performance. In extreme cases, assertions in the reference TFLITE interpreter built in to TensorFlow may fail, causing TensorFlow to abort without providing a meaningful error-status return.

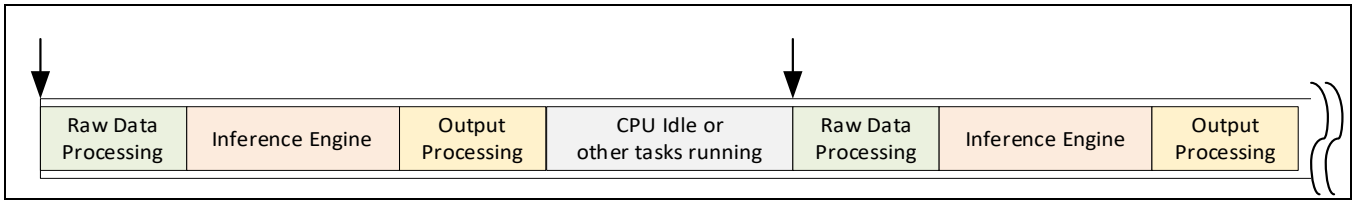
7.3 Memory and CPU requirements

When using the TFLM inference engines, different types of memory blocks are required, and special attention is required to allocate them. The following table provides a summary of what is required at a minimum:

Memory block	Location	Description
Model weights	Flash or SRAM	Contains the model weights. It can be very large, requiring to be stored in the external memory.
Tensor arena	SRAM	A combination of scratch and persistent memory, handle automatically by the TLFM inference engine
Input Buffer	SRAM	Buffer storing the input data to the NN. It depends on how many nodes in the input layer. Some types of NN have a huge number of inputs, for example, when processing images.
Output Buffer	SRAM	Buffer storing the output data to the NN. It depends on how many nodes in the output layer. There is a hard limit of 64 nodes in the output layer.
Regression Input Data	Flash	Only used for regression. Usually it is very large and require to be stored in external flash. With the ML 3.x solution, we recommend to stream regression data.
Regression Output Data	Flash	Only used for regression. Usually it is very large and require to be stored in external flash. With the ML 3.x solution, we recommend to stream regression data.

ModusToolbox™ ML integration

In terms of CPU requirements, it is important to understand what the sample rate is, the sensor processing time, the inference engine time and output processing time. The following graphic shows an example how the CPU handles the bandwidth on every task.



The total amount of time it takes to do all the processing around the data needs to be smaller than the period of the sampling rate. Choosing a different quantization for the inference engine can drastically reduce the cycles time, at the cost of using more memory. See the Evaluating a Model section.

Revision history

Revision	Date	Description
**	2021-03-12	New tool.
*A	2021-04-29	Added information for the dataset structures. Updated table of files generated by the ML Configurator.
*B	2021-08-18	Updated instructions based on ModusToolbox™ ML 1.2 solution.
*C	2021-09-14	Add information about the "Advanced scratch memory optimization" check box in the ML Configurator. Clarified the note about the array with the model weights, and how to add to internal flash and external memory.
*D	2022-07-15	Updated instructions based on ModusToolbox™ ML 2.0 solution. Added support for TensorFlow Lite for Microcontrollers. Added more information to the ml-coretools section. Added list of layers/operators coverage supported by TFLite and Keras formats Added JSON field parameters table for MTBML format Updated list of files generated by the ML Configurator
*E	2022-08-23	Updated getting started instructions. Added note about TFLM inference using int8x8 quantization. Added information about the sparsity feature.
*F	2022-10-10	Added instructions to install QEMU Added note about warning messages when generating/validating the model
*G	2022-11-10	Updated links to Infineon website to download/install the pack. Updated QEMU installation instructions.
*H	2023-05-23	Explained the usage of the mtb_ml_utils_quantize() function. Removed support for 2D and 3D feature data for CSV format. Added instructions to enable CMSIS_DSP component.
*I	2023-11-02	Removed support for Infineon inference engine. Added references to Imagimob and ML application note. Added support for the PSOC™ Edge MCU.
*J	2023-12-18	Added a new section “Validating a model” Added more information on cycle count estimation Added more information on the CSV format
*K	2024-10-15	Removed CMSIS component dependencies Added Vela compiler options information Added instructions for PSOC Edge to model placement
*L	2025-01-23	Added support to int16x8 quantization Added support to NNLITE Added table of supported operators on NPU Added cache optimization option for the CM55+U55
*M	2025-01-27	Minor edits.
*N	2025-04-23	Added a new section to list the cache management types for Ethos-U55 Added more info on the files generated by the ML-configurator
*O	2025-09-24	Explained that ML_ INT8x8 / ML_INT16x8 / ML_FLOAT32 are now optional Fixed DEEPCRAFT code examples links; Added the AI kits references

Revision history

Revision	Date	Description
*P	2026-02-11	Updated with Keras 3 information. Added note about Vela-processed tflite models for the Configurator.
*Q	2026-03-23	Update with multi-input and multi-output support information.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2026-03-23

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2026 Infineon Technologies AG. All Rights Reserved.

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

002-32590 Rev. *Q

Important notice

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”)

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer’s technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies’ products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.